Parallel 2D Gray-Scott Simulation using CUDA

Erik Pahor, Jaka Škerjanc

I. Introduction

This report presents a parallel implementation of the 2D Gray-Scott reaction-diffusion model using CUDA C++. The Gray-Scott system simulates complex pattern formation arising from the interaction of two chemical species. Our goal was to significantly accelerate this computationally demanding simulation by leveraging GPU parallelism. We developed sequential C and parallel CUDA versions, employing finite differences, forward Euler integration, and periodic boundary conditions. The CUDA version was optimized using shared memory for efficient data access. Performance was benchmarked on the Arnes HPC cluster across various grid sizes and model parameters, with a focus on execution time and speedup.

II. Implementation Approach

A. Gray-Scott Model and Discretization

The simulation solves the Gray-Scott equations (Eqs. for $\partial U/\partial t, \partial V/\partial t$ omitted for brevity due to standard formulation) on an $N \times N$ grid. A 5-point stencil approximates the Laplacian ∇^2, and the forward Euler method updates concentrations U, V over discrete time steps Δt. Periodic boundary conditions wrap grid edges.

B. Sequential and Parallel Implementations

The sequential C version computes updates cell-by-cell, using double buffering for U and V grids to maintain data integrity within each time step.

The parallel CUDA version assigns each grid cell update to a GPU thread. **Shared Memory Tiling:** Thread blocks load tiles of U and V (including a 1-cell halo for neighbors) into shared memory. This minimizes global memory reads for the stencil computation, as threads within a block access the faster shared memory. Periodic boundary conditions are handled during this loading phase. **Double Buffering on GPU:** Two sets of device memory buffers are used for U and V, with pointers swapped after each kernel, avoiding device-to-device copies. **Minimized Host-Device Transfers:** Data is transferred to the GPU once initially and results are copied back once at the end.

III. Experimental Setup

Tests were run on the Arnes cluster. Sequential code used *gcc* with $-O3$; CUDA code used *nvcc* with $-O3$, C++11, targeting sm_70/sm_80. Fixed parameters: $\Delta t = 1.0$, $D\_u = 0.16$, $D\_v = 0.08$. Benchmarks ran for 5000 steps on grids 256^2 to 4096^2. Five (F, k) parameter sets were tested: Default (0.060, 0.062), Flower (0.055, 0.062), Mazes (0.029, 0.057), Mitosis (0.028, 0.062), and Solitons (0.030, 0.060). Timing via *clock_gettime* included H-D/D-H transfers for CUDA.

A. Thread Block Size Optimization

Prior to full benchmarking, the optimal thread block size for the CUDA kernel was investigated using the $N = 4096$ grid for 10000 steps with Default (F,k) parameters. Results are in Table I.

Table I: Execution Time (s) for $N = 4096$, 10000 steps, varying block sizes.

Block Size (X×Y)	CUDA Time (t_p)
8×8	8.894 765
16×16	7.780 832
32×16	7.595 705
16×32	7.959 354
32×32	7.967 386

The 32×16 (512 threads/block) configuration yielded the best performance and was used for all subsequent benchmarks presented in Table II.

IV. Results and Discussion

Performance results are summarized in Table II. Figure 2 shows an example static pattern. To illustrate the dynamic evolution, selected frames from a color-mapped video simulation ($N = 256$, 'Default' pattern) are presented in Figure 3.

Table II: Execution Times (s) and Speedup (S) for 5000 steps (Block Size 32×16).

Grid Size	Pattern	Seq Time (t_s)	CUDA Time (t_p)	Speedup (S)
256^2	Default	3.65	0.028	128.56
	Flower	3.78	0.028	135.13
	Mazes	3.67	0.028	130.00
	Mitosis	3.78	0.028	134.16
	Solitons	3.57	0.028	127.27
512^2	Default	12.38	0.061	202.76
	Flower	12.50	0.061	204.18
	Mazes	12.51	0.061	206.40
	Mitosis	12.52	0.060	207.13
	Solitons	12.13	0.061	199.41
1024^2	Default	65.05	0.271	240.00
	Flower	64.58	0.272	237.06
	Mazes	56.39	0.276	204.55
	Mitosis	59.55	0.268	222.52
	Solitons	58.91	0.264	223.39
2048^2	Default	258.33	0.982	263.16
	Flower	260.66	0.974	267.62
	Mazes	265.66	0.977	271.82
	Mitosis	269.35	0.984	273.69
	Solitons	261.86	0.986	265.58
4096^2	Default	867.52	3.80	228.29
	Flower	868.47	3.81	227.95
	Mazes	872.45	3.80	229.59
	Mitosis	896.48	3.79	236.54
	Solitons	876.30	3.80	230.61

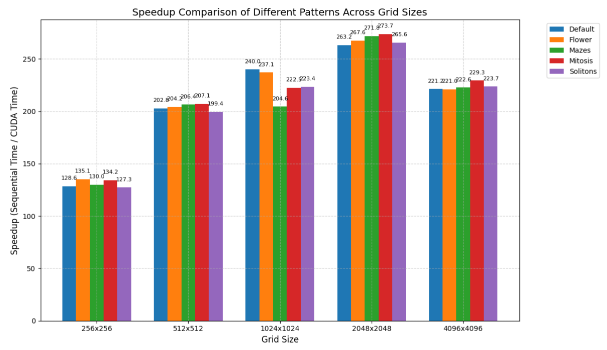

Figure 1: Speedup of CUDA vs. Sequential C (averaged over patterns).

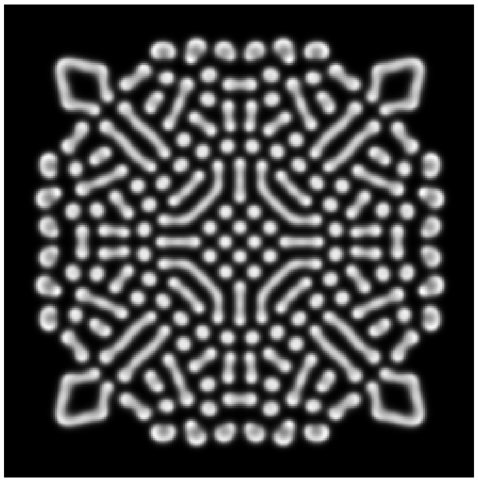

Figure 2: Species V for 'Solitons' (F=0.03, k=0.06), $N = 256$, 5000 steps (CUDA).

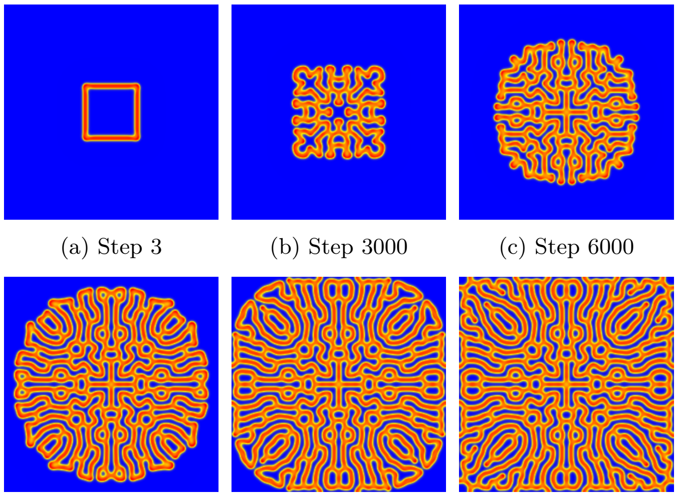

Figure 3: Selected frames from a color video simulation of the 'Default' pattern (F=0.060, k=0.062) on a 256×256 grid, showing the evolution of species V concentration over time. Colors map V concentration (e.g., blue for low, red for high).

The CUDA implementation yields substantial speedups, exceeding $200\times$ for $N \geq 512$. Speedup generally increases with grid size up to $N = 2048$, as larger grids better utilize GPU parallelism and amortize overheads. The use of shared memory is critical for performance by reducing global memory latency.

The thread block size optimization (Table I) showed 32×16 (512 threads) as optimal for $N = 4096$. This configuration likely balances parallelism, register usage, and shared memory capacity per SM effectively for this problem size on the target GPU. The slight speedup decrease observed previously at $N = 4096$ with the 32×16 configuration. Different (F,k) parameters had minimal impact on performance, confirming that stencil computations dominate runtime.

V. Conclusion

The CUDA-accelerated 2D Gray-Scott solver achieved significant speedups (up to $\approx 270\times$ with 32×16 blocks) over a sequential C implementation. Optimizing thread block size and leveraging shared memory were key to this performance. GPUs are highly effective for such stencil-based simulations. Further work could involve dynamic block size selection or exploring multi-GPU scaling.